DATABASE
The Journal of Biological Databases and Curation

Database, 2019, 1–10
doi: 10.1093/database/baz096
Review

Review

Benchmarking database systems for Genomic Selection implementation

Yaw Nti-Addae[1,*], Dave Matthews[2], Victor Jun Ulat[3], Raza Syed[1], Guilhem Sempéré[4], Adrien Pétel[5], Jon Renner[6], Pierre Larmande[7], Valentin Guignon[8], Elizabeth Jones[1] and Kelly Robbins[9]

[1]Institute of Biotechnology, Cornell University [2]Boyce Thompson Institute [3]Centro Internacional de Mejoramiento de Maíz y Trigo (CIMMYT) [4]INTERTRYP, Univ Montpellier, CIRAD, IRD [5]UMR PVBMT, CIRAD [6]University of Minnesota [7]UMR DIADE, IRD, University of Montpellier [8]Bioversity International [9]Section of Plant Breeding and Genetics, School of Integrative Plants Sciences, Cornell University

*Corresponding Author Email Address: yn259@cornell.edu

Citation details: Nti-Addae,Y., Matthews,D., Ulat,V. J. *et al.* Benchmarking database systems for Genomic Selection implementation. *Database* (2019) Vol. 2019: article ID baz096; doi:10.1093/database/baz096

Received 26 February 2019; Revised 29 May 2019; Accepted 1 July 2019

Abstract

Motivation: With high-throughput genotyping systems now available, it has become feasible to fully integrate genotyping information into breeding programs. To make use of this information effectively requires DNA extraction facilities and marker production facilities that can efficiently deploy the desired set of markers across samples with a rapid turnaround time that allows for selection before crosses needed to be made. In reality, breeders often have a short window of time to make decisions by the time they are able to collect all their phenotyping data and receive corresponding genotyping data. This presents a challenge to organize information and utilize it in downstream analyses to support decisions made by breeders. In order to implement genomic selection routinely as part of breeding programs, one would need an efficient genotyping data storage system. We selected and benchmarked six popular open-source data storage systems, including relational database management and columnar storage systems.

Results: We found that data extract times are greatly influenced by the orientation in which genotype data is stored in a system. HDF5 consistently performed best, in part because it can more efficiently work with both orientations of the allele matrix.

Availability: http://gobiin1.bti.cornell.edu:6083/projects/GBM/repos/benchmarking/browse

Introduction

The development of Next Generation Sequencing (NGS) technologies has made it feasible to generate huge volumes of genomic data. In the field of plant breeding, the availability of cost-effective genomic data has the potential to change the way crop breeding is done. Genomic selection (GS) is a breeding method where the performance of new plant varieties is predicted based on genomic information (1). Multiple studies have shown the potential of this methodology to increase the rates of genetic gain in breeding programs by decreasing generation interval, the time it takes to screen new offspring and identify the best performers for use as parents in the next generation (2, 3). Although model capabilities exist for the implementation of GS, mainstream applications require the computational infrastructure to manage NGS data, often generated from highly multiplexed sequencing runs that generate low coverage data with large amounts of missing information. The lack of computational infrastructure remains a major barrier to the routine use of genomic information in public sector breeding programs.

The Genomic Open-source Breeding informatics initiative (GOBii) is a project aimed at increasing the rates of genetic gain in crop breeding programs serving regions in Africa and South Asia by developing the capabilities required for routine use of genomic information. To deal with the technical challenges of storage, rapid access (i.e. query execution), and computation on NGS data, the initial focus of GOBii has been the development of an efficient genomic data management system (GOBii-GDM). The system must be able to efficiently store huge volumes of genomic information and provide rapid data extraction for computation. The system must be scalable for large breeding programs while being able to run effectively at institutions with limited access to large computational clusters. While many open-source technologies exist for the management of large two-dimensional datasets, it is unclear which technologies best suit the needs of plant breeding and genetics research.

There are many appealing characteristics of traditional relational database management systems (RDBMS), which are designed and built to store, manage, and analyze large-scale data. However, performance can be problematic when dealing with large matrix data like those commonly encountered in genomic research. One common limitation of RDBMS is database partitioning, which allows for a logical database to be divided into constituent parts and distributed over a number of nodes, (e.g. in a computer cluster) (4). To address these performance issues, many RDBMS have capabilities for working with binary large objects (BLOBs). Current versions of PostgreSQL (version 9.3 and up) have support for JSONB objects that could be used to store BLOBs of genomic data. However, this still does not solve the data retrieval performance issues (4). An alternative to storing genomic data directly in a RDBMS is to use a hybrid system (5) with high-dimensional data being stored in files and key meta information required for querying the data stored in an RDBMS.

Leveraging several decades of work of the database community on optimizing query processing, columnar store databases such as MonetDB are designed to provide high performance on complex queries against large databases, such as combining tables with hundreds of columns and millions of rows. In the biological context, experiences show that MonetDB (8) enables the data to be stored in a format to allow fast queries of vectors of genomic data based on marker or sample indexes, which should improve performance relative to RDBMS. More recently NoSQL systems have emerged as effective tools for managing high-dimensional genomic data (9–11). NoSQL systems for distributed file storage and searching represent scalable solutions comparable to RDBMS when dealing with semi-structured data types (12, 13), and MongoDB, a document-based NoSQL database, has been used to develop a web-based tool for exploring genotypic information (14).

The Hierarchical Data Format (HDF5) is a member of the high-performance distributed file systems family. It is designed for flexible, efficient I/O and for high-volume and complex data. It has demonstrated superior performance with high-dimensional and highly structured data such as genomic sequencing data (6) making it an appealing option for a hybrid system approach. There are an increasing number of bioinformatics applications, such as BioHDF (20), SnpSeek (7), Oxford Nanopore PoreTools (21) and FAST5, all of which use HDF5 to scale up simple execution on large numbers of documents. However, there is little reported information on the performance of HDF5 when the system is used to process more complex analytical queries that involve aggregations and joins.

To determine the ideal technology to serve as the back-end of the GOBii-GDM, testing was performed using a large genotype-by-sequencing (GBS) dataset (15, 16, 19). Open-source RDBMS, PostgreSQL and MariaDB, a community-developed fork under the GNU GPL of MySQL, were used as a baseline for performance testing and compared with HDF5, MonetDB, Elasticsearch (17), Spark (18), and MongoDB. Loading and extraction times were measured using queries that would be commonly run for applications of GS in a breeding program.

Methods

Six database systems were tested using a subset of a maize nested association mapping (NAM) population GBS

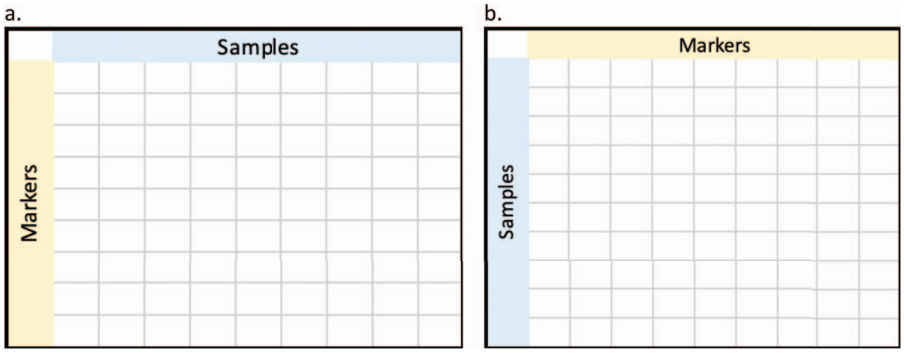

Figure 1. Different orientation of genotyping data. (a) markers in rows and samples in columns, whereas (b) shows markers in columns and samples in rows.

SNP dataset (19) containing allele calls for 5258 samples (germplasm lines) and 31,617,212 markers. Each marker represents a physical position in the reference genome where polymorphisms were detected in the samples. Each genotyping call was encoded as a single ASCII character for the diploid result, using IUPAC nucleotide ambiguity codes for heterozygotes and "N" for missing data. The input and output format for all tests was a text file containing only the tab-delimited allele calls.

Genomic matrices can be stored in two different orientations as shown in Figure 1. Given that traditional RDBMS (PostgreSQL and MariaDB) or disk stores are optimized for extracting data by rows, and columnar stores (MonetDB and Parquet) are optimized for extracting data by columns, we stored data in these two different orientations where possible for each of the database systems. Unfortunately, due to size and technology restrictions, not all systems could handle both orientations. We tested if the orientation of genotype matrix has an impact on query execution times. For the purposes of this benchmarking, we will define "marker-fast" orientation as the orientation of the genotype matrix in a system that favors the extract of markers, and converse, "sample-fast" as orientation of data in a system that favors the extract of samples. For example, in MonetDB, sample-fast orientation will have samples in columns and markers in rows and as such favor the querying of samples to markers. Vice versa, sample-fast orientation in PostgreSQL will have samples in rows and markers as indexes in a binary JSON object.

Three use cases were used to test the performance of systems with queries set up to extract data by:

I. All samples for a list of markers (USECASE I). We will refer to this use case as MARKER-EXTRACT USECASE throughout the remainder of this article.
II. All markers for a list of samples (USECASE II). We will refer to this use case as SAMPLE-EXTRACT USECASE throughout the remainder of this article.
III. A block of data defined by a list of markers and samples (USECASE III). We will refer to this use-case as BLOCK-EXTRACT USECASE.

Table 1. Server configuration and software versions

Server Configuration	
Processor	Intel Xeon E5 2620 V3 2.4GHz 6C 15 Mb
Number of Processors	24
Memory	128GB DDR-42133 MHz ECC/REG
Raid	6
Network File Storage	50 TB
Operating System	Debian
Software	**Version**
HDF5	1.8
MariaDB	10.1
MonetDB	1.1
MongoDB	3.6.0 (compression level: snappy)
PostgreSQL	9.5
Spark	2.3
Java Hotspot	1.8 (64 bit)
Elasticsearch	2.3.3 (under Java 1.8.0_92 64bit)

For each use case, we tested extracting a contiguous list of markers or samples versus a random list. Care was necessary to avoid unrepeatable timing results due to memory caches in the system. We turned off caching features of database systems where applicable to ensure repeated test runs were consistent with first-time execution of queries on each system. Reported times are an average of execution times for multiple runs of the same query. All tests were run on a single node server with a 10 gigabit ethernet connection to a fast-access file server. The server specifications and version of the tested systems are listed in Table 1.

Database Implementation

The default parameters were applied to each benchmarked system, with some parameters critical for performance, such as memory allocation, manually optimized.

Table 2. Data stored in "marker-fast" orientation in PostgreSQL and MariaDB. Marker column is of type string and contains the name of the marker, and Data column is of type JSON object for PostgreSQL and Dynamic column for MariaDB, and contains a JSON formatted string

Marker	Data
S6_120018	{"B73(PI550473):250028951":"C","B97(PI564682):250027958":"T","CML103(Ames27081):250027883":"C","CML228(Ames27088):250027960":"C"}
S6_120046	{"B73(PI550473):250028951":"A","B97(PI564682):250027958":"A","CML103(Ames27081):250027883":"A","CML228(Ames27088):250027960":"A"}

PostgreSQL Implementation

PostgreSQL is one of the most widely used open source object-relational database systems. Tests were done in version 9.5 and its configuration file was modified to consume up to 30GB of memory per query, whereas the default configurations use only 250 MB of memory. Data was stored in "marker-fast" orientation for MARKER-EXTRACT USECASE as shown in Table 2.

Conversely, data was stored in sample-fast orientation for SAMPLE-EXTRACT USECASE, where samples were in one column and associated markers and SNP allele calls in JSON format in another column. We recognize that PostgreSQL can be optimized in cases of highly sparse data by storing only non "N" SNPs, which allows PostgreSQL to act like a document store, thereby reducing the size of stored information by many fold. For the purposes of this benchmarking, all data was treated as complete as many use cases in genomic selection will pull information in which missing data has been imputed.

MariaDB Implementation

MariaDB is a community-developed, commercially supported fork of MySQL relational database. MariaDB was configured to utilize up to the maximum available memory on the server. Similar to PostgreSQL, data was stored in marker-fast for MARKER-EXTRACT USECASE, as shown in Table 2, and sample-fast for SAMPLE-EXTRACT USECASE. Although the data format is the same as in PostgreSQL, MariaDB uses a Dynamic column data object to store the JSON string in Data field. Similar to JSONB object columns in PostgreSQL, Dynamic columns in MariaDB allow one to store different sets of columns, in a JSON format, for each row in a table. It works by storing a set of columns in a BLOB and having a small set of functions to manipulate it.

MongoDB Implementation

MongoDB is an open source document-oriented database system. MongoDB tests were performed using version 3.6.0 configured with the WiredTiger storage engine and the snappy compression level, a choice driven by a comparison work done in (14). MongoDB was tested with data stored in both orientations, marker-fast and sample-fast. For the marker-fast orientation, two types of documents were used:

- one for storing the genotype array corresponding to each marker, as follows:

```
{
    "_id": "S6_120018",
    "g": [.
      "T",
      "A",
      "C",
      "N", ...]
}
```

- the second for mapping sample names to indices in the latter array.

```
{
  "_id": "CML247(PI595541):250027894",
  "n": NumberInt(0).
}
```

In the sample orientation, the sample collection remained the same as above, but the documents storing genotypes were refactored as:

```
{
  "_id": {.
    "ii": "B73(PI550473):250028951",
    "c": NumberInt(0).
  },
  "g": [.
    "C",
    "A",
    "G",
    "C", ...]
}
```

Ideally, the document id would have been just the sample name, and the genotype array length would have been equal to the number of markers, i.e. 31,617,212. But because MongoDB has a 16 Mb document-size limitation, we had to split each sample's genotype array into chunks of maximum size 100,000. This explains why the id here is composite and consists in the following pair: sample name + chunk index.

Table 3. Times in minutes for extracting M markers for all samples

Contiguous	# of markers (mil)	MonetDB	HDF5	MongoDB	PostgreSQL	MariaDB	Spark
Yes	0.001	0.1	0	0	0.1	1566	19.5
Yes	0.005	0.2	0	0.1	0.5	1623	43.45
Yes	0.01	0.3	0	0.3	0.8	1523	71.3
Yes	0.05	1.2	0.1	1.4	3.9	1579	373.08
Yes	0.1	3.7	0.3	2.5	7.8	1663	423.07
Yes	0.5	10	1.2	12.3	39.2	1520	420.32
Yes	1	21.7	2.3	26.8	78.4	1632	426.8
Yes	5	134.2	11.6	134.3	391.8		475
No	0.001	44.5	0.1	0.2	0.5	1545	252
No	0.005	47.2	0.4	0.7	0.6	2043	251
No	0.01	47.8	0.9	1.8	1.2	2108	251
No	0.05	58.6	4.4	7.1	6	3017	252
No	0.1	74.8	8.6	13.7	13.2	6021	252
No	1	2700	80.8	130.8	149.5		264
No	5		443	858.8	945.6		316

Table 4. Times in minutes for extracting N samples for all markers

Contiguous	# of samples	MonetDB	HDF5	MongoDB	PostgreSQL	MariaDB	Spark
Yes	50	1.8	2	190.8	1282	1830	2.7
Yes	100	4	4	195.8	1301	1868	2.82
Yes	500	42.1	19.6	234.4	1401		13.73
Yes	1000	103.2	40.5	291.4	1528		32.63
Yes	2000	238.5	84.7	405.5	1804		92.52
Yes	3000	432.5	128.4	518.2	2069		149.38
Yes	4000	698.5	169	632.8	2327		211.52
No	50	1.9	1.7	188.5	1482	1575.63	2.9
No	100	4.7	2.4	195.6	1505	1535.53	3.3
No	500	48.7	15.6	239.3	1648		14
No	1000	118.3	22.6	295.7	1696		35.5
No	2000	306.6	58.5	409	1678		108
No	3000	547	81.6	621.3	1674		171
No	4000	626.9	107.7	677.9	1681		239

HDF5 Implementation

Hierarchical Data Format (HDF5) file format is designed to store and organize extremely large and complex data collections. The allele matrix was stored in one-byte cells in both orientations, marker-fast with samples as columns (HDF5 dimension 1) and markers as rows (dimension 0) and sample-fast in the opposite orientation. When extracting data for a list of samples and a list of markers, BLOCK-EXTRACT USECASE, the most straightforward approach would be to use HDF5's *H5Sselect_elements* function to address each allele call by its (sample, marker) coordinates. However, it was found to be much faster to extract all the data for each marker into a memory array using *H5Sselect_hyperslab*, and then look up the results for the desired samples in that array. The speed increase from this approach was more than 30-fold under all conditions tested.

Under some conditions HDF5 performance can be improved by structuring the HDF5 data file in a "chunked" format, depending on the patterns of data access in actual use. A chunk is a contiguous two-dimensional block of the data array that is stored and retrieved as a unit whenever any element within it is accessed. The results reported above were obtained without chunking. Separate tests were performed to compare the unchunked format with four different (marker, sample) chunk dimensions: (256, 256), (1000, 1000), (1000, 5258), and (1, 5258). Loading the HDF5 file was no faster for any of these configurations, and 7-fold slower for (1000, 1000). Retrieval times were also not improved by chunking, using the *H5Sselect_hyperslab* procedure described above.

Spark Implementation

Apache Spark is an open-source distributed general-purpose cluster computing framework with an in-memory data processing engine. Spark version 2.3.0 was used, together with PySpark (Python 3.6.5) interface. The

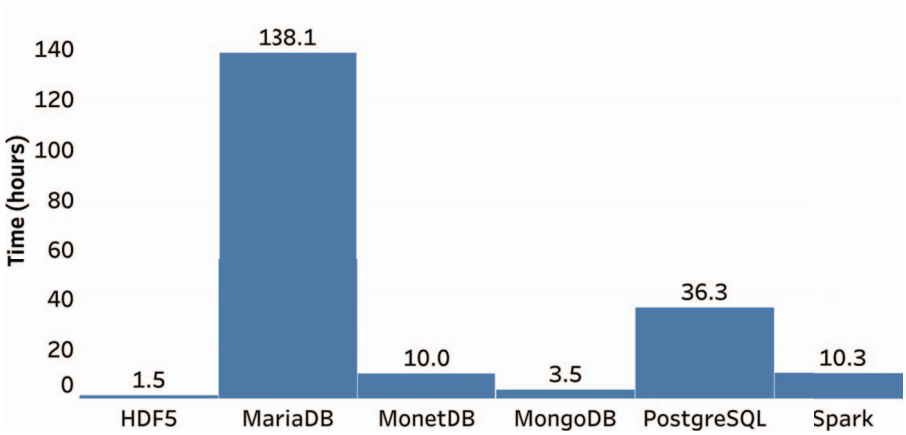

Figure 2. Load times for database systems.

Table 5. Times in minutes for extracting M markers by N samples

Contiguous?	# of markers	# of Samples	MonetDB	HDF5	MongoDB	PostgreSQL	MariaDB	Spark
Yes	1k	1000	0.00	0.0	0.00	0.10	1287	1.60
Yes	1k	3000	0.10	0.0	0.00	0.10	1684	8.55
Yes	1k	4000	0.10	0.0	0.00	0.10	1718	13.37
No	1k	1000	8.40	0.1	0.20	0.20	1073	20.4
No	1k	3000	28.60	0.1	0.20	0.20	1163	144
No	1k	4000	24.70	0.1	0.20	27.10	1662	201
Yes	50k	1000	0.10	0.10	0.70	2.60	1726	38.08
Yes	50k	3000	0.40	0.10	1.20	3.40	1635	204.3
Yes	50k	4000	0.80	0.20	1.50	3.80	1111	299.3
No	50k	1000	10.80	4.10	6.60	5.00	1297	20.05
No	50k	3000	33.20	4.20	7.20	5.20	1623	145
No	50k	4000	32.40	4.20	8.00	4.80	1756	203
Yes	1m	1000	3.40	0.90	12.10	49.90	1045.00	41.32
Yes	1m	3000	10.60	2.10	22.10	68.20	1048.00	222.12
Yes	1m	4000	14.70	2.60	29.60	76.10	1492.00	314.2
No	1m	1000	192.1	72.6	109.60	121.80	1802.00	22.77
No	1m	3000	1846.6	78.5	138.10	143.40	1911.00	152.5
No	1m	4000	3291.7	83.0	153.90	151.00	1935.00	212.6

Java VM used was Java Hotspot 64 bit 1.8.0_171. Although Spark operates optimally in a distributed cluster environment (23), we set it up in its simplest form as a standalone node cluster to expose Spark to the same amount of CPU and memory as the other database systems. For data storage, we implemented Apache Parquet, which is a columnar storage file format available to any project in the Hadoop ecosystem like Hive, Impala and Pig, and has an interface for most programming languages. For this exercise we used PySpark. It is possible there might be a small improvement in benchmark results using the pure Scala interface for Spark, though in general the overhead for using PySpark is not large. Spark benchmarks were run by reading from a pre-prepared Parquet file-format version of the genotype matrix. Data was stored in sample-fast orientation with samples in columns and markers in rows, and vice versa for marker-fast orientation.

MonetDB Implementation

MonetDB is an open source column-oriented database management system. Version 1.1 of MonetDB was used for the benchmarking. Similar to Parquet file format in Spark, data was stored in sample-fast orientation with samples as columns and markers in rows. Due to the column number restriction in MonetDB, we were not able to store data in marker-fast orientation as the number of markers exceeded the limit for number of columns. MonetDB out of the box is configured to use any available memory on the server.

Elastic search Implementation

To achieve optimal performance, ES settings must be tuned according to each dataset, hardware environment and expected response time. Furthermore, ES is not designed to

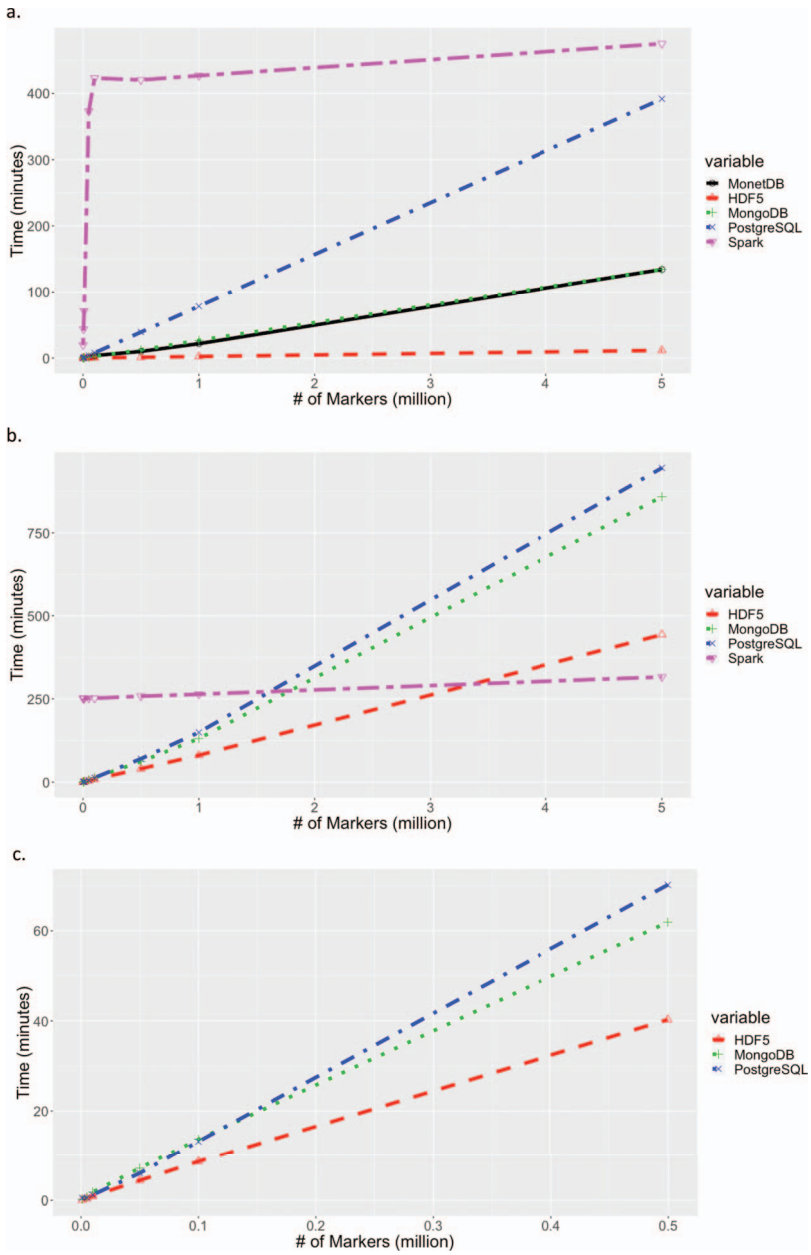

Figure 3. Times for extracting increasing number of markers across all samples. (a) Times for extracting a contiguous set of markers for all samples. Times for MariaDB are excluded because they exceed 25 hours, and times for MongoDB and MonetDB were essentially identical. (b) Times for extracting a random set of markers for all samples. Times for MariaDB and MonetDB are excluded since they exceed 25 hours. (c) A zoom-in at extract times up to 500,000 random markers to show if there is significant difference between HDF5, MongoDB and PostgreSQL.

return large amounts of data at once, using a scrolling API instead, which complicates the task of gathering query results. Based on these initial results and the complexity of implementing ES as part of GOBii-GDM solution, the decision was made to not pursue further benchmarking on ES.

Results and Discussion

Data Loading

The load times for each system are presented in Figure 2. HDF5 was the fastest, with MongoDB also performing reasonably well. The two RDMS performed poorly, with MariaDB being the worst, taking approximately 90 times longer than HDF5. While loading time is a lower priority than extraction times, the tight turnaround times from receiving marker data from the lab and generating genomic predictions for selection makes loading times of more than a day for large datasets undesirable for routine implementation of GS. While the process used for loading the data was not optimized, it is unlikely that loading times for MariaDB could be reduced to an acceptable level of performance. The performance of PostgreSQL could potentially be reduced to

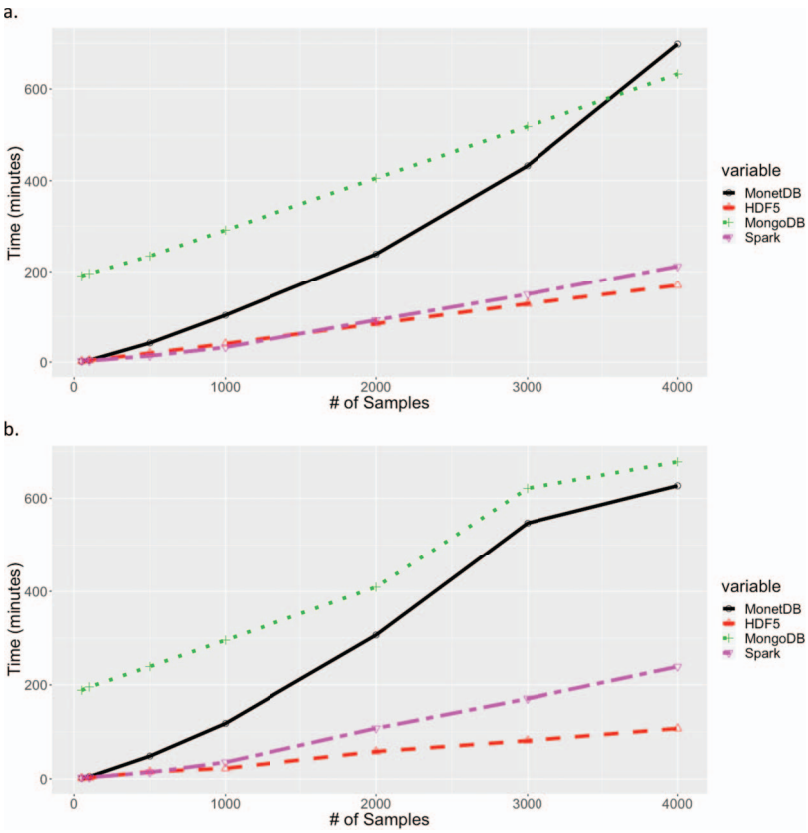

Figure 4. Times for extracting increasing number of samples across all markers. (a) Times for extracting contiguous set of samples for all 32 million markers. (b) Times for extracting random set of samples for all 32 million markers. Times for MariaDB and PostgreSQL are excluded in both (a) and (b) because their queries exceeded 20 hours.

less than 1 day, but the large gap in performance compared to the others is undesirable. Overall, given that none of the systems have been optimized, we find HDF5, MonetDB and Spark loading times to be acceptable for routine GS implementation.

Data Extraction

Data extraction tests intentionally extended to very large result sets to test performance beyond the limits expected in actual use. Figure 3 shows the extraction times for increasing number of markers, for all samples in the dataset. For a contiguous block of markers, HDF5 showed the best performance, with the next best solution, MonetDB, performing 11 times slower. On the other hand, for a random list of markers, although HDF5 shows better performance overall, Spark showed a steady performance across different marker blocks, and seems to outperform HDF5 at high marker numbers. The big discrepancy in performance of Spark between contiguous and random list of markers can be explained in the columnar nature of Spark. Spark Parquet file format is a column-oriented data format, so it does not have an "index" of rows and their order, so to ask for a contiguous chunk of rows is antithetical to the design of the data format. Results for MariaDB were greater than 25 hours for all points, off scale in Figure 3a and 3b. When selecting random markers for all samples, MonetDB extraction times exceeded 25 hours for even modest numbers of markers. This is likely due to the orientation of the data stored in the system. Due to limitations in the number of columns for the MonetDB and Spark set-up used for benchmarking, data could not be stored with markers as the columns (> 31 million columns). Given that MonetDB is designed for fast querying of columns, the system did not perform well for marker (row) queries. For lower numbers, less than 500,000, of random markers, a practical case in genomic selection, Figure 3b shows that there might not be significant difference in performance between HDF5, MongoDB and PostgreSQL. Figure 3c zooms in on fewer random markers and shows that HDF5 is more than 1.5 times faster than both MongoDB and PostgreSQL.

Extraction times for retrieving all markers for subsets of samples are shown in Figure 4. Again, the results for MariaDB were greater than 25 hours for all queries. As expected MonetDB performed significantly better with queries on samples (columns). As with queries on markers, HDF5

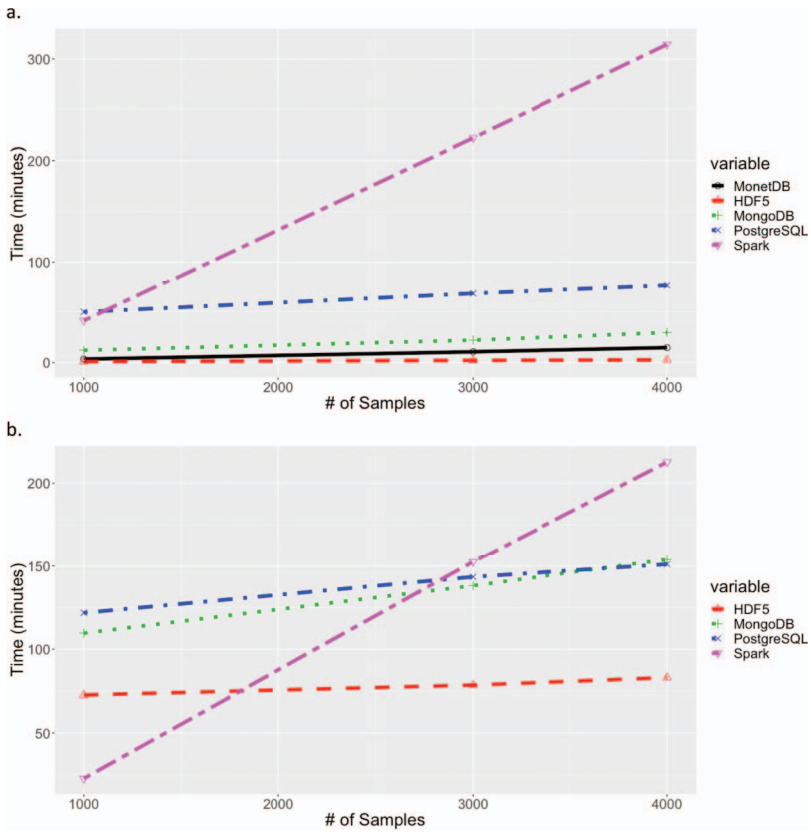

Figure 5. Times for extracting increasing cross-section of samples and 1 million markers. Time for extracting a block contiguous number of samples and 1 million contiguous markers. Extract times for MariaDB are excluded since they exceeded 25 hours. (b) Times for extracting random set of samples across 1 million random markers. Extract times for MariaDB and MonetDB are excluded because their queries exceeded 25 hours.

performed best, with the next best solution, Spark, being 1.2 and 2.2 times slower than HDF5 for the contiguous and random sample scenarios respectively. For queries on samples, the relative performance of PostgreSQL dropped substantially, with extract times exceeding 20 hours for all data points. The drop in relative performance, and insensitivity to number of samples in the random sample scenario, may be related to the way in which the data is stored in PostgreSQL. A JSONB object is stored for each marker, with each object containing the allele calls for all samples. The observed performance indicates that the total extraction time is influenced more strongly by the time to fetch the JSONB objects from disk into memory, than to extract data for desired samples from the JSONB objects in memory. For the queries extracting all markers from selected samples, two data orientations were tested for MongoDB, with the sample-fast orientation (shown) performing significantly better.

Results from BLOCK-EXTRACT USECASE, extraction based on varying lists of markers and samples. Results for a list of 1 million markers and varying numbers of samples is shown in Figure 5. Once again HDF5 gave the best performance with the next best system performing 5.7 and 1.8 times slower for the largest contiguous and random marker and sample lists, respectively. For the contiguous scenario, MonetDB gave the second-best performance, twice as fast as MongoDB, but exceeded 48 hours for the random list selecting data from 4000 samples and 1 million markers. Neither of the RDBMS performed well for scenario 3, with all extractions taking more than 48 hours for 4000 samples and 1 million markers.

Conclusion

For all USECASEs the orientation of the data storage had a substantial impact on the performance time of extraction. The fact that PostgreSQL and MonetDB had limitations on storing the data in sample fast and marker fast orientations reduces their utility for systems that would be regularly queried based on either markers or samples. For systems using HDF5 or MongoDB, best performance would be obtained by storing the data in both orientations with queries being directed to the optimal orientation. While HDF5 showed consistently superior performance in extraction times and loading, implementation in a genomic data management system would require a hybrid

approach, with critical meta information likely stored in a RDBMS. A final determination on whether to build a system around HDF5 would need to account for the performance and complexity of developing and deploying a hybrid system. All performance tests were done using a fixed number of cores, but previous studies have shown that the performance of NoSQL distributed file systems, such as MongoDB and Spark, increase with access to more cores (8). Further testing is required to determine if the performance of MongoDB or Spark would surpass HDF5 when deployed on large clusters.

Funding

This work has been supported by the Bill & Melinda Gates Foundation and Agropolis Foundation grant E-SPACE (1504–004).

Conflict of interest. None declared.

References

1. Meuwissen,T.H.E., Hayes,B.J. and Goddard,M.E. (2001) Prediction of Total Genetic Value Using Genome-Wide Dense Marker Maps, *Genetics*, **157**, 1819 LP-1829.
2. Hickey,J.M., Chiurugwi,T., Mackay,I. *et al.* (2017) Genomic prediction unifies animal and plant breeding programs to form platforms for biological discovery. *Nat. Genet.*, **49**, 1297.
3. Lin,Z., Hayes,B.J. and Daetwyler,H.D. (2014) Genomic selection in crops, trees and forages: a review, *Crop Pasture Sci.*, **65**, 1177–1191.
4. Wang,S. *et al.* (2014) High dimensional biological data retrieval optimization with NoSQL technology, *BMC genomics*, **15**, S3.
5. Röhm,U. and Blakeley,J. (2009) Data management for high-throughput genomics, *arXiv Prepr*. arXiv0909.1764.
6. Hoffman,M.M., Buske,O.J. and Noble,W.S. (2010) "The Genomedata format for storing large-scale functional genomics data," *Bioinformatics*, **26**, 1458–1459.
7. Alexandrov,N., *et al.* (2014) "SNP-Seek database of SNPs derived from 3000 rice genomes," *Nucleic Acids Res.*, **43**, D1023–D1027.
8. Cijvat,R. *et al.* (2015) "Genome sequence analysis with MonetDB," *Datenbank-Spektrum*, **15**, 185–191.
9. Guimaraes,V. *et al.* (2015) "A study of genomic data provenance in NoSQL document-oriented database systems," in Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on, 1525–1531.
10. Manyam,G., Payton,M.A., Roth,J.A. *et al.* (2012) "Relax with CouchDB—Into the non-relational DBMS era of bioinformatics," *Genomics*, vol. **100**, no. 1, pp. 1–7.
11. Gabetta,M., Limongelli,I., Rizzo,E. *et al.* (2015) "BigQ: a NoSQL based framework to handle genomic variants in i2b2," *BMC Bioinformatics*, **16**, p. 415.
12. Schulz,W.L., Nelson,B.G., Felker,D.K. *et al.* (2016) Evaluation of relational and NoSQL database architectures to manage genomic annotations. *J. Biomed. Inform.*, **64**, 288–295.
13. Dede,E., Govindaraju,M., Gunter,D. *et al.* (2013) "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pp. 13–20.
14. Sempéré,G., Philippe,F., Dereeper,A. *et al.* (2016) "Gigwa—Genotype investigator for genome-wide analyses," *Gigascience*, **5**, 25.
15. McMullen,M.D. *et al.* (2009) "Genetic properties of the maize nested association mapping population," *Science (80-.)*, **325**, 737–740.
16. Glaubitz,J.C. *et al.* (2014) "TASSEL-GBS: a high capacity genotyping by sequencing analysis pipeline," *PLoS One*, **9**, e90346.
17. Gormley,C. and Tong,Z. (2015) *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. O'Reilly Media, Inc.
18. Zaharia,M. *et al.* (2016) "Apache spark: a unified engine for big data processing,"*Commun. ACM*, **59**, 56–65.
19. Bukowski,R., Guo,X., Lu,Y. *et al.* (2018) Construction of the third-generation Zea mays haplotype map, *GigaScience*, **7**, gix134, https://doi.org/10.1093/gigascience/gix134.
20. Mason,C.E. *et al.* (2010) Standardizing the Next Generation of Bioinformatics Software Development with BioHDF (HDF5). In: Arabnia H (ed). *Advances in Computational Biology. Advances in Experimental Medicine and Biology*, Vol. **680**. Springer, New York, NY.
21. Loman,N., Quinlan,A. Poretools: a toolkit for analyzing nanopore sequence data, bioRxiv 007401; doi: https://doi.org/10.1101/007401.
22. Thomson,M.J. (2014) "High-Throughput SNP Genotyping to Accelerate Crop Improvement.".
23. Ahmed *et al.* (2016), "Performance Comparison of Spark Clusters Configured Conventionally and a Cloud Service".